Justin Sim and Mina Gao (Lab Group 4)
EE 371
February 13, 2024
Lab 4 Report

Procedure

In this lab, we are tasked with implementing algorithms as hardware circuits by using ASMD charts. This lab consists of two tasks. For task 1, we built a bit-counting circuit, designed to count the number of '1' bits in an 8-bit input. In the second task, we created a binary search algorithm, which can locate a given value within sorted arrays efficiently.

Task #1

In Task 1, we focused on implementing a bit-counting circuit using an ASMD chart as a guideline. The primary objective was to count the number of '1' bits in an 8-bit input, provided through slide switches on the LabsLand DE1-SoC board. For the design, we incorporated a Finite State Machine (FSM) for control and datapath components like a shift register and a counter. The FSM managed the process flow, transitioning between states for loading data, counting bits, and indicating completion. The shift register was key for handling the input data, while the counter tracked the number of '1' bits. The circuit's output was displayed on a 7-segment display and an LED, providing a clear indication of the counted bits and the completion of the process.

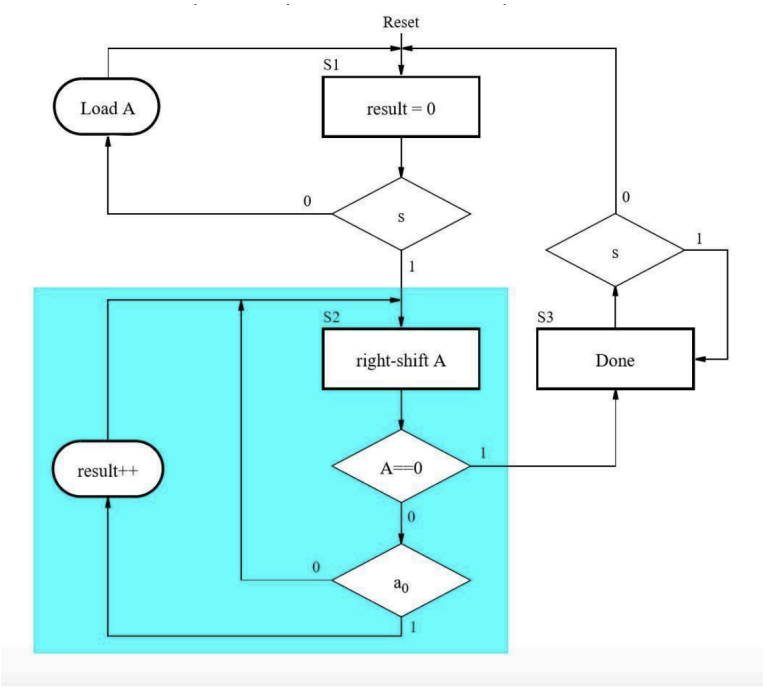

Figure 1: ASMD chart for Task 1

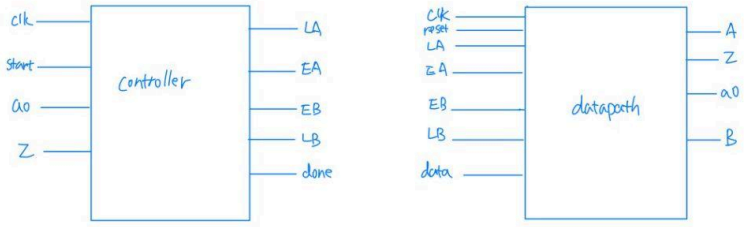

Figure 2: Block Diagram for Task 1

Task #2

For Task 2, we are asked to implement a binary search algorithm, aiming to locate an 8-bit value in a sorted array. The array is stored in a memory module within the FPGA chip using a 32x8 RAM, and the value to be searched is specified using switches. The design, implemented based on the ASMD chart shown below, comprises a Finite State Machine for controlling the search process and a datapath for handling data comparisons and address calculations. The circuit produces a 5-bit address output, displayed on 7-segment displays, which indicates the location of the searched value in the array. LEDs 9 and 8 are used to signal whether the value was found or not, enhancing user interaction. The difficulty with this task was working around the delay issue in the datapath between the algorithm and the RAM. We end up using a DFF to intentionally delay all the other signals to run the algorithm at the same clock as the RAM.

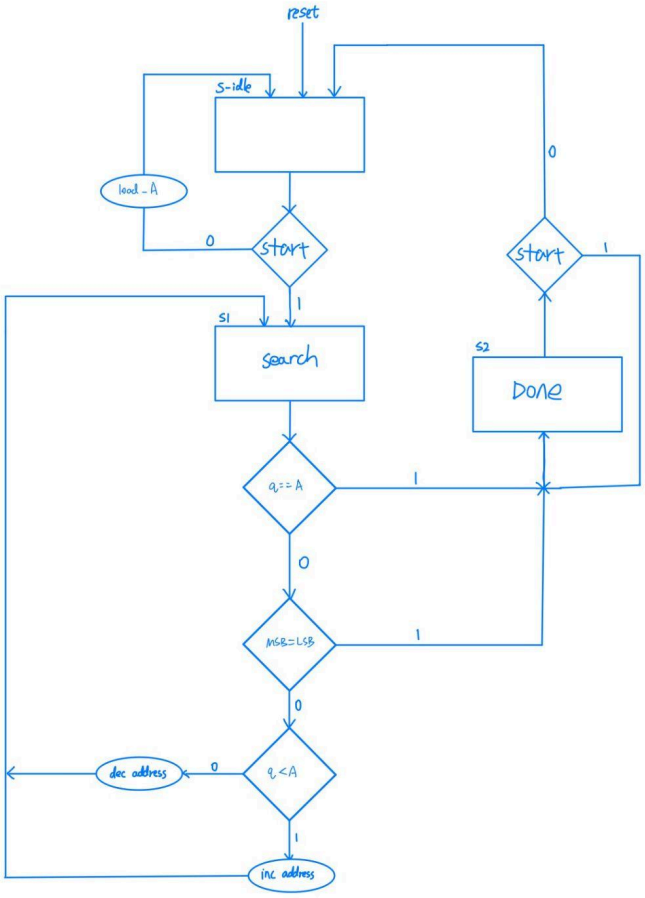

Figure 3: ASMD chart for Task 2

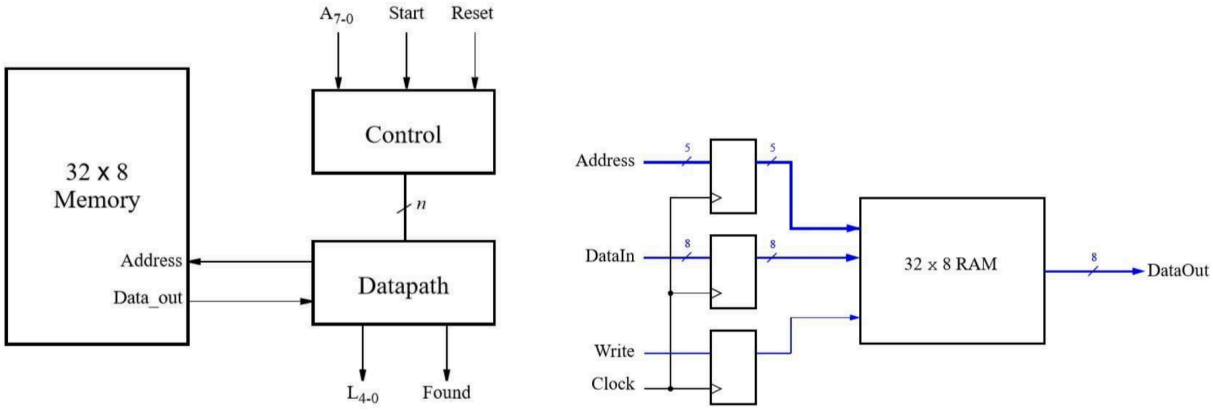

Figure 4: Block Diagram for Task 2

Results

Task 1: datapath_tb():

Inputs:

1) input logic clk,
2) input logic reset,
3) input logic LA,
4) input logic EA,
5) input logic LB, // Load Counter
6) input logic EB, // Enable Counter
7) input logic [n-1:0] data,

Outputs:

1) output logic [n-1:0] A,
2) output logic Z, a0,
3) output logic [$clog2(n)-1:0] B); // Count (3'b)

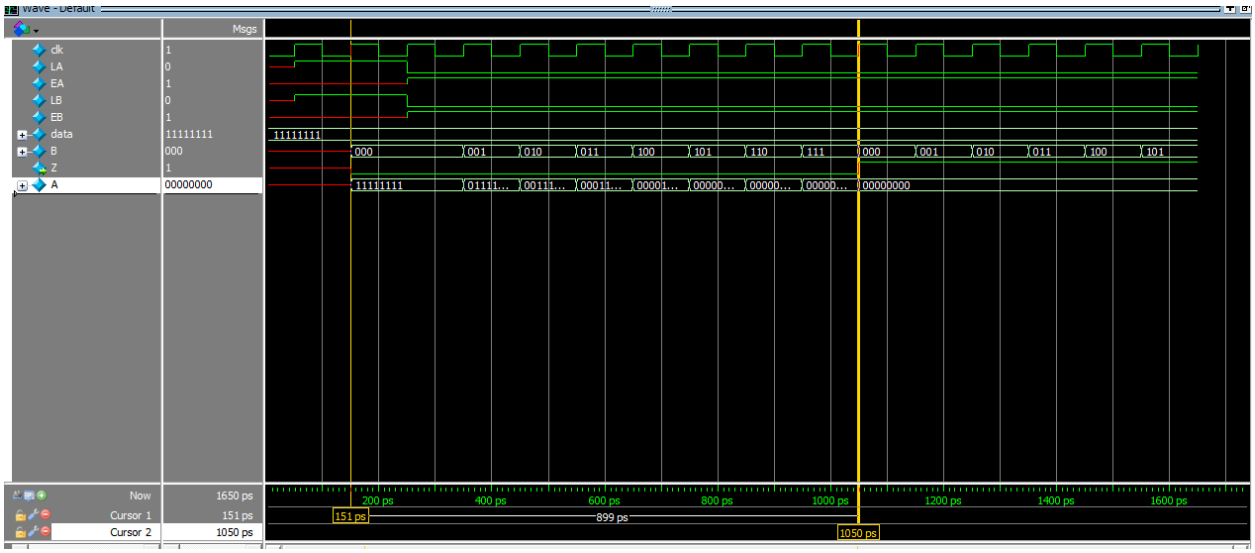

Figure 5: datapath testbench ModelSim

The module above shows the testbench for datapath in Task 1. We are testing the functionality when the data input is all 0s, we should see that the output A is shifting the input to the right each time and replacing the leftmost digit with a 1. This is simulating the shifter module. Additionally, we have the counter module, signified by the B. At the end (1050 ps) we can see that B reaches its maximum value meaning that it correctly counted all the 1s in the inp[ut data stream.

Task 1: controller_tb():

Inputs:

1) input logic clk, reset,
2) input logic start,
3) input logic a0, // least significant bit
4) input logic Z, // 1 if signal is dead

Outputs:

1) output logic LA, // load A
2) output logic EA, // enable A
3) output logic EB, // enable B
4) output logic LB, // load B
5) output logic done

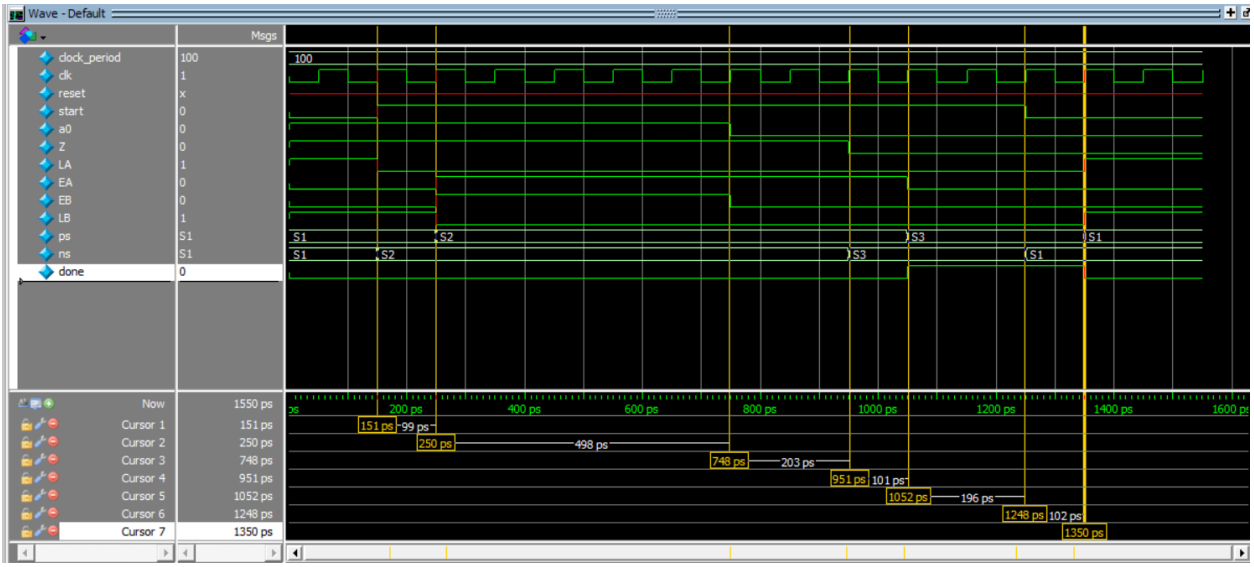

Figure 6: controller testbench for task 1

The figure above shows the controller module testbench for task 1. We can verify that it is doing the expected behaviors. The behaviors we are trying to achieve are:

1) Go to each state (S1, S2, S3)
2) Have start on while in state S3

- We can see that it visits State S1 at the start when reset is high, as expected (151 ps)
- We can see that it visits S2 when EA is high, as expected, because S2 is where the algorithm begins (250 ps)
- It visits S3 at (748 ps) when EA is low, as expected because this occurs when the algorithm is done.
- At 1052, the module is reading S3, while we have started control high, according to our ASMD, we know that it should stay in S3, as it does.

Task 1: DE1_SoC_testbench():

Inputs:

1) input logic [9:0] SW,
2) input logic [3:0] KEY,
3) input logic CLOCK_50);

Outputs:

1) output logic [6:0] HEX0,
2) output logic [9:0] LEDR,

Figure 7: DE1_SoC testbench for task 1

The figure above shows the DE1_SoC top level module for Task 1. This module should show the following parameters hold

1) A_in consists of 4 1's
2) B = 4 at the end

We can see that B is increasing whenever a0 (the leftmost digit of A) is high. This means that B is keeping track of the number of 1s in the input data stream A. We can also see that A is shifting to the right while 0s are filling its place on the right, as expected from our shifter in the datapath. Lastly, the reset functionality at the end is effectively making B = 0, ready for the next stream of data. All in all, this testbench effectively showed us that the program is effective in counting the number of 1s in the data stream, implementing both the controller and datapath modules, and it is ready for the board.

HexadecimalConverter_testbench():

Depending on the 4-bit input, it will return the corresponding Hexadecimal character, which was helpful for Task 1 where we wanted to convert binary input to something readable on the HEX display.

Inputs:

1) clk
2) [3:0] binary_input

Output:

1) [6:0] hexadecimal_output

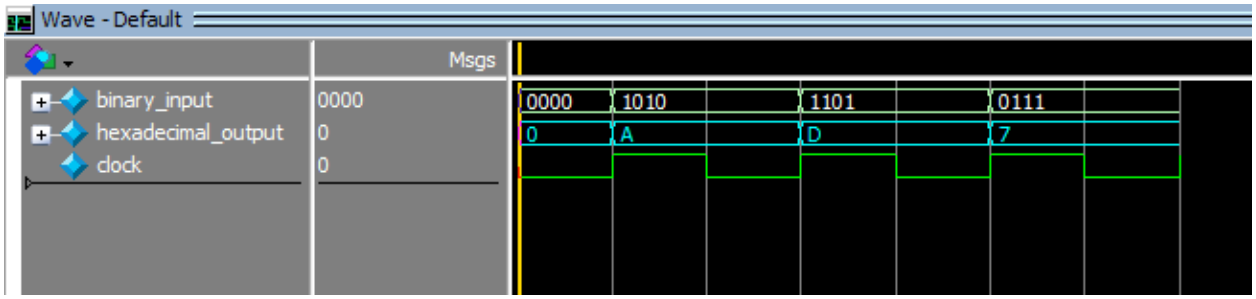

Figure 8: HexadecimalConverter testbench Modelsim

The image above shows a testbench for the module. A radix setting was set following the structure of a common 7-bit hexadecimal conversion (0 to F). The intended behavior is to display the selected characters (A,D,7) for one clock cycle each. It is easy to see that the testbench is aligned with our expectations, meaning that this module can be implemented to perform 4-bit digit to 7-bit hexadecimal conversion.

Task 2: DE1_SoC_testbench():

Inputs:

1) CLOCK_50
2) [3:0] KEY;
    a) KEY0: Reset signal
3) [9:0] SW;
    a) SW9: start signal
    b) SW7-0: specify the value A

Outputs:

1) [6:0] HEX;
    a) HEX1-0: display the address of A if found
2) [9:0] LEDR;
    a) LEDR9: Found
    b) LEDR8: Not Found

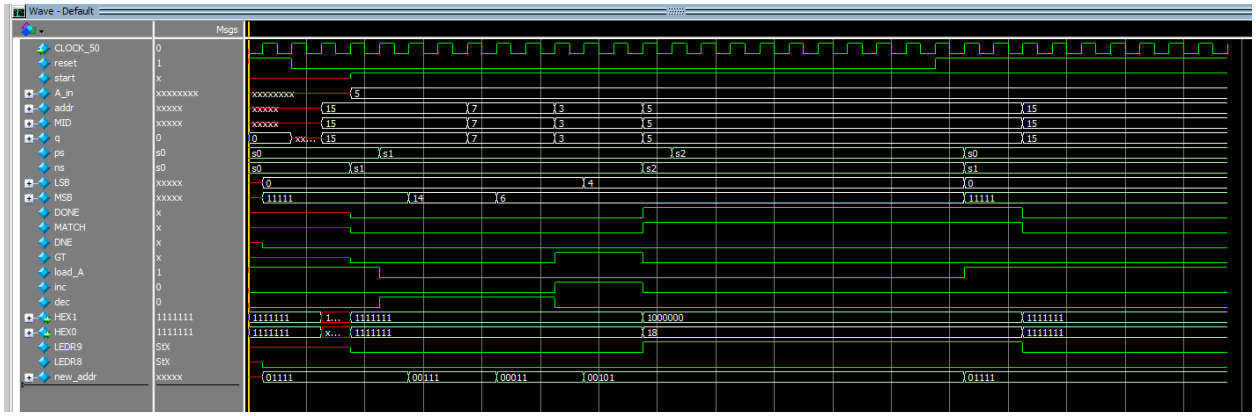

Figure 9: DE1_SoC testbench for task 2

The image above shows a testbench for the top module, which is testing if we can successfully find an existing value 5 in the memory using the binary search, after combining the fsm and datapath modules.

1) reset for 2 clock cycles (s0), load A
2) then set start signal to high
3) then going in to search state (s1)
4) after found a match, go to done state (s2)
5) HEX display is activated showing the address of A, and LED9 lights up
6) hold the value until reset

According to the simulation, we can verify that the module successfully demonstrates the binary search. And it is ready to be tested on the DE1-SoC board.

Task 2: binary_search_FSM_testbench():

Inputs:

1) clk
2) reset
3) start
4) MATCH
5) DNE
6) GT
7) eq

Outputs:

1) load_A
2) inc
3) dec

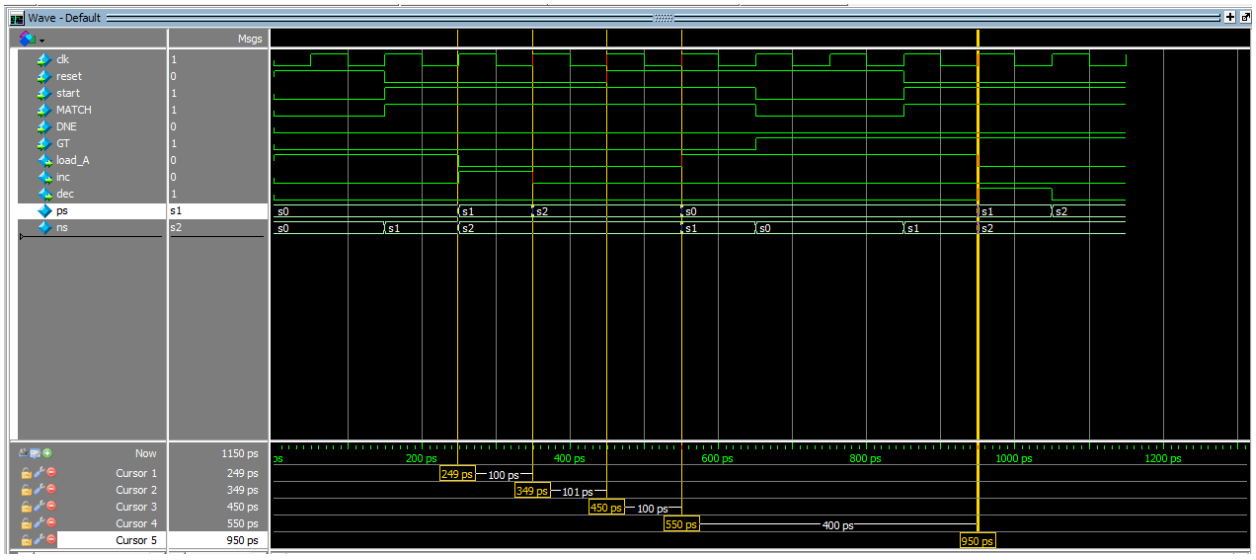

Figure 10: fsm_testbench for Task 2

The image above shows a testbench for the binary search FSM module. We can verify that it is doing the expected behaviors. The behaviors we are trying to achieve are:

1) Go to each state (s0, s1, s2)
2) Verify the transition between states

- We can see that it is at State s0 and loading A at the beginning when reset is high and goes to s1 when the start signal is on, as expected.
- We can see that it goes to s2 when MATCH is high, as expected, because if we find a match the search algorithm is in the done state.
- After we set the reset signal to high again, it goes back to s0.

The states are getting updated as expected, which means it can successfully implement the binary search algorithm.

Task 2: binary_search_datapath_testbench():

Inputs

1) clk
2) [7:0] A_in
3) load_A
4) inc
5) dec

Outputs:

1) [4:0] addr
2) MATCH
3) DNE

4) GT
5) DONE
6) Eq
7) [7:0] q

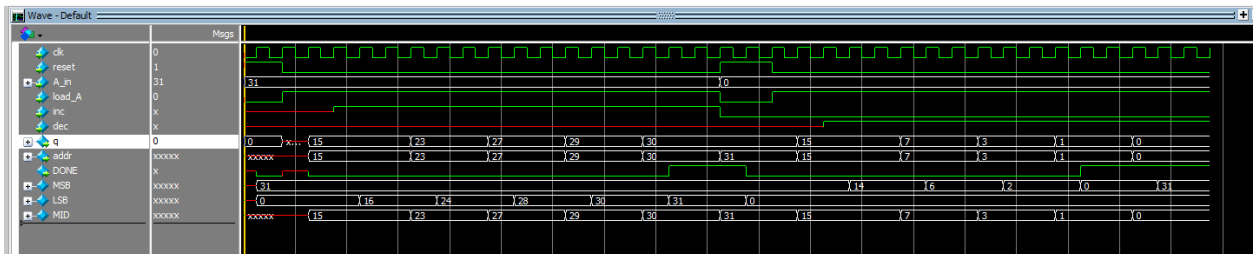

Figure 11: binary_search_datapath_testbench Modelsim

The figure above shows the expected behavior of datapath. We are testing the following two properties:

1) a _in = 31
2) A_in = 0

By testing these two signals, we can see if the datapath is effectively going through the binary search algorithm. According to the figure above, we can see that the algorithm is correctly doing what we predicted it to. DONE is set to HIGH when MSB == LSB, indicating that the algorithm is done. This happens because the DONE is set to the end of the algorithm.

Task 2: Hex7_testbench():

Depending on the 5-bit input (designed to take the 5-bit output 'addr' from the binary search datapath), it will return the corresponding Hexadecimal character, which was helpful for Task 2 where we wanted to convert binary input to something readable on the HEX display.

Input:

1) [4:0] count

Outputs:

1) [6:0] HEX4 (hexadecimal output)
2) [6:0] HEX5 (hexadecimal output)

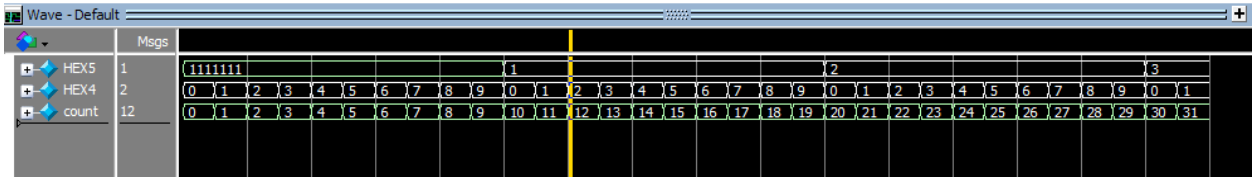

Figure 12: Hex7_testbench Modelsim

The image above shows a testbench for the module. As we updated count, HEX4 and 5 got updated accordingly which aligned with our expectations, meaning that the module can perform 5-bit binary to 7-bit hexadecimal conversion.

Appendix

```
1   /*
2   Justin Sim and Mina Gao
3   2/5/2024
4   EE 371 Hussein
5   Lab 4, Task 1
6
7      Datapath Module:
8         waits for control signals from controller and
9         returns data shifted by 1, and the final count/
10        NO. of 1s in the signal
11
12     inputs:
13        All SW               SW7-0 use for data / SW9 use for start
14        KEY0                 KEY0 use for reset
15        CLOCK_50             clk
16     outputs:
17        [6:0] HEX:           7'b HEX display to show count
18        LEDR9                use to indicate done
19
20  */
21  module DE1_SoC #(parameter n = 8)
22     (output logic [6:0] HEX0,
23     output logic [9:0] LEDR,
24     input logic [9:0] SW,
25     input logic [3:0] KEY,
26     input logic CLOCK_50);
27
28     logic clk;
29     logic reset;
30     logic done;
31     logic start;
32     logic a0;
33     logic LA;
34     logic EA;
35     logic LB;
36     logic EB;
37     logic [n-1:0] data;
38     logic [n-1:0] A;
39     logic Z;                        // 1 if signal shifted signal is 0
40     logic [$clog2(n)-1:0] B;           // stores count
41
42     assign clk = CLOCK_50;
43     assign reset = ~KEY[0];
44     assign LEDR[9] = done;
45     assign start = SW[9];
46     assign data = SW[7:0];
47
48     // Instantiate Datapath
49     datapath dp(.*);
50
51     // Instantiate controller
52     controller ctrl(.*);
53
54     // Display the Count
55     HexadecimalConverter_3b disp(.clk, .binary_input(B), .hexadecimal_output(HEX0));
56
57  endmodule                          // DE1_SoC
58  /*-------------------------------------------------------------------------------
59  DE1_SoC_tb testbench for expected and unexpected cases */
60
61  module DE1_SoC_tb();
62     parameter n = 8;
63     logic [6:0] HEX0;
64     logic [9:0] LEDR;
65     logic [9:0] SW;
66     logic [3:0] KEY;
67     logic CLOCK_50;
68     logic reset;
69     logic done;
70     logic start;
71     logic a0;
72     logic LA;
73     logic EA;
```

```
74      logic LB;
75      logic EB;
76      logic [n-1:0] data;
77      logic [n-1:0] A;
78      logic Z;                        // 1 if shifted signal is 0
79      logic [$clog2(n)-1:0] B;        // stores count
80
81      assign reset = ~KEY[0];
82      assign done = LEDR[9];
83      assign start = SW[9];
84      assign data = SW[7:0];
85
86      DE1_SoC dut(.*);
87
88      ///// Testbench Clock Setup/////
89      parameter clock_period = 100;
90      initial begin
91         CLOCK_50 <= 0;
92         forever #(clock_period /2) CLOCK_50 <= ~CLOCK_50;
93      end
94
95
96      initial begin
97         KEY[0] <= 1;
98         SW[7:0] = 8'b10101010;
99         SW[9] <= 0;
100        @(posedge CLOCK_50);
101        @(posedge CLOCK_50);
102        SW[9] <= 1;
103        @(posedge CLOCK_50);
104        @(posedge CLOCK_50);
105        @(posedge CLOCK_50);
106        @(posedge CLOCK_50);
107        @(posedge CLOCK_50);
108        @(posedge CLOCK_50);
109        @(posedge CLOCK_50);
110        @(posedge CLOCK_50);
111        @(posedge CLOCK_50);
112        @(posedge CLOCK_50);
113        @(posedge CLOCK_50);
114        KEY[0] <= 0;
115        @(posedge CLOCK_50);
116        @(posedge CLOCK_50);
117        $stop;
118     end
119
120  endmodule                          // DE1_SoC_tb
```

```
1   /*
2   Justin Sim and Mina Gao
3   2/5/2024
4   EE 371 Hussein
5   Lab 4, Task 1
6
7      Controller Module:
8         provides sequential logic for datapath
9         determines when to shift input signal
10        and when to add to the result
11
12     inputs:
13        clk            clock
14        reset          reset the counter
15        start          starts the module
16        A              8-bit input from SW[7:0]
17        a0             least significant bit of current A
18     outputs:
19        Load_A         calls the datapath to load the input signal
20        result         final count of number of 1's in A
21        right_shift_A  tells the shift register to shift input signal A
22
23  */
24  module controller(
25     input logic clk, reset,
26     input logic start,
27     input logic a0,                     // least significant bit
28     input logic Z,                      // 1 if signal is dead
29     output logic LA,                    // load A
30     output logic EA,                    // enable A
31     output logic EB,                    // enable B
32     output logic LB,                    // load B
33     output logic done
34  );
35     /* S1: initial state, result = 0
36        S2: occurs when start = 1, increments result to count no. of 2's in reg A
37        S3: occurs when A = 0, sets output to done
38     */
39
40
41     enum {S1, S2, S3} ps, ns;
42
43     always_comb begin
44        case(ps)
45           S1: begin
46              if (start) begin          // start
47                 ns = S2;
48              end else begin
49                 ns = S1;
50              end
51           end
52
53           S2: begin
54              if (Z) begin
55                 ns = S3;               // signal ends
56              end else begin
57                 ns = S2;
58              end
59           end
60
61           S3: begin
62              if (start) begin               // case where start in done state
63                 ns = S3;
64              end else begin
65                 ns = S1;               // wait for next signal
66              end
67           end
68        endcase
69     end                                 // always_comb
70
71     always_ff @(posedge clk) begin
72        if (reset)
73           ps <= S1;
```

```
74          else
75             ps <= ns;
76       end
77
78       // outputs
79       assign LA = (ps == S1) & (start == 0);
80       assign LB = (ps == S1);
81       assign EA = (ps == S2);
82       assign EB = (ps == S2) & (a0 == 1);
83       assign done = (ps==S3);
84
85    endmodule                                  // controller
86    /*-------------------------------------------------------------------------------
87    controller_tb testbench for all 3 states of the controller module */
88
89    module controller_tb();
90       logic clk, reset, start, a0, Z, LA, EA, EB, LB, done;
91
92
93       controller dut(.*);
94
95       ///// Testbench Clock Setup/////
96       parameter clock_period = 100;
97       initial begin
98          clk <= 0;
99          forever #(clock_period /2) clk <= ~clk;
100      end
101
102      initial begin
103         Z <= 1;
104         a0 <= 1;          // signal's LSB is 1
105         start <= 0;
106         @(posedge clk);   // Load A
107         @(posedge clk);
108         start <= 1;
109         @(posedge clk);   // S2
110         @(posedge clk);
111         @(posedge clk);
112         @(posedge clk);
113         @(posedge clk);   // result = 1
114         @(posedge clk);   // result = 2
115         a0 <= 0;          // result = 2
116         @(posedge clk);
117         @(posedge clk);
118         Z <= 0;
119         @(posedge clk);   // S3
120         @(posedge clk);
121         @(posedge clk);   // start is still true
122         start <= 0;
123         @(posedge clk);   // S1
124         @(posedge clk);
125               @(posedge clk);
126         $stop;
127      end
128
129   endmodule                                  // controller_tb
130
```

```
1   /*
2   Justin Sim and Mina Gao
3   2/5/2024
4   EE 371 Hussein
5   Lab 4
6
7      Datapath Module:
8         waits for control signals from controller and
9         returns data shifted by 1, and the final count/
10        NO. of 1s in the signal
11
12     inputs:
13        clk            clock
14        reset          reset the counter
15        LA
16        EA
17        LB
18        EB
19        [7:0] data     input signal (size n)
20     outputs:
21        Z              True if Signal A !=0
22        [2:0] B        Final count No. of 1s (size log2(n))
23
24  */
25  module datapath #(parameter n = 8)
26     (input logic clk,
27     input logic reset,
28     input logic LA,
29     input logic EA,
30     input logic LB,               // Load Counter
31     input logic EB,               // Enable Couner
32     input logic [n-1:0] data,
33     output logic [n-1:0] A,
34     output logic Z, a0,
35     output logic [$clog2(n)-1:0] B);             // Count
36
37
38
39     // counter
40     always_ff @(posedge clk) begin
41        if (reset || LB)
42           B <= 0;
43        else if (EB && B !=3'b111)          // prevent circling back to 0
44           B++;
45        else if (B==3'b111)
46           B <= 3'b111;
47     end
48
49     // shift register
50     always_ff @(posedge clk) begin
51        if (LA)
52           A <= data;
53        else if (EA)
54  //       A <= {1'b0, A[n-1:1]};  // board
55           A <= {0, A[n-1:1]};  // simulation
56     end
57
58     assign Z = ~|A;
59     assign a0 = A[0];
60
61  endmodule                        // datapath
62  /*-------------------------------------------------------------------------------
63  datapath_tb testbench shows counter, shifter, and Z=1 when A is all 0 */
64
65  module datapath_tb();
66     parameter n = 8;
67     logic clk;
68     logic reset;
69     logic LA;
70     logic EA;
71     logic LB;               // Load Counter
72     logic EB;               // Enable Couner
73     logic [n-1:0] data;
```

```
74      logic Z;
75      logic [$clog2(n)-1:0] B;               // Count
76      logic a0;
77      logic [n-1:0] A;
78
79      datapath dut(.*);
80      ///// Testbench Clock Setup/////
81      parameter clock_period = 100;
82      initial begin
83         clk <= 0;
84         forever #(clock_period /2) clk <= ~clk;
85      end
86
87      initial begin
88         data = 8'b11111111;
89         @(posedge clk);
90         LA = 1;
91         LB = 1;
92         @(posedge clk);   // Load A
93         @(posedge clk);
94         LA = 0;
95         LB = 0;
96         EA = 1;
97         EB = 1;
98         @(posedge clk);   // Count and Shift
99         @(posedge clk);
100        @(posedge clk);
101        @(posedge clk);
102        @(posedge clk);
103        @(posedge clk);
104        @(posedge clk);
105        @(posedge clk);
106        @(posedge clk);
107        $stop;
108     end
109
110  endmodule                              // datapth_tb
111
```

```
1   module HexadecimalConverter_3b (clk, binary_input, hexadecimal_output);
2      input logic [2:0] binary_input;
3      input logic clk;
4      output logic [6:0] hexadecimal_output;
5      logic [6:0] h0, h1, h2, h3, h4, h5, h6, h7;
6
7      assign h0 = 7'b1000000; // 0
8      assign h1 = 7'b1111001; // 1
9      assign h2 = 7'b0100100; // 2
10     assign h3 = 7'b0110000; // 3
11     assign h4 = 7'b0011001; // 4
12     assign h5 = 7'b0010010; // 5
13     assign h6 = 7'b0000010; // 6
14     assign h7 = 7'b1111000; // 7
15
16    always_comb begin
17      case (binary_input)
18        3'b000: hexadecimal_output = h0;
19        3'b001: hexadecimal_output = h1;
20        3'b010: hexadecimal_output = h2;
21        3'b011: hexadecimal_output = h3;
22        3'b100: hexadecimal_output = h4;
23        3'b101: hexadecimal_output = h5;
24        3'b110: hexadecimal_output = h6;
25        3'b111: hexadecimal_output = h7;
26
27      endcase
28    end
29
30  endmodule
31
32
33  module HexadecimalConverter_3b_testbench;
34
35    logic [2:0] binary_input;
36    logic [6:0] hexadecimal_output;
37    logic clock;
38
39    // Instantiate HexadecimalConverter module
40    HexadecimalConverter_3b dut (
41      .binary_input(binary_input),
42      .hexadecimal_output(hexadecimal_output)
43    );
44
45    // clock setup
46     parameter clock_period = 100;
47     initial begin
48        clock <= 0;
49        forever #(clock_period / 2) clock <= ~clock;
50     end
51
52    // Test stimulus
53    initial begin
54      binary_input = 3'b0000;       @(posedge clock);
55
56      // Apply test vectors
57      binary_input = 3'b010;     @(posedge clock);//
58      binary_input = 3'b101;     @(posedge clock);//
59      binary_input = 3'b111;     @(posedge clock);//
60
61      // Add more test vectors as needed
62
63     $stop; // Stop simulation after a certain duration
64    end
65
66  endmodule
67
```

```
module Hex7(count, HEX5, HEX4);
   input logic [4:0] count;
   output logic [6:0] HEX5, HEX4;

   //for 0-9 and A-F
   logic [6:0] h0, h1, h2, h3, h4, h5, h6, h7, h8, h9, hOff;
   //logic [6:0] hA, hB, hC, hD, hE, hF;

   assign h0 = 7'b1000000; // 0
   assign h1 = 7'b1111001; // 1
   assign h2 = 7'b0100100; // 2
   assign h3 = 7'b0110000; // 3
   assign h4 = 7'b0011001; // 4
   assign h5 = 7'b0010010; // 5
   assign h6 = 7'b0000010; // 6
   assign h7 = 7'b1111000; // 7
   assign h8 = 7'b0000000; // 8
   assign h9 = 7'b0010000; // 9
   assign hOff = 7'b1111111;  // no display

   //for letters (A-F)
   //assign hA = 7'b0001000; // A
   //assign hB = 7'b0000011; // B
   //assign hC = 7'b1000110; // C
   //assign hD = 7'b0100001; // D
   //assign hE = 7'b0000110; // E
   //assign hF = 7'b0001110; // F


   always_comb begin
      case(count)
         // 0
         5'b00000:
            begin
                  HEX5 = hOff;
                  HEX4 = h0;
            end

         // 1
         5'b00001:
            begin
                  HEX5 = hOff;
                  HEX4 = h1;
            end

         // 2
         5'b00010:
            begin
                  HEX5 = hOff;
                  HEX4 = h2;
            end

         // 3
         5'b00011:
            begin
                  HEX5 = hOff;
                  HEX4 = h3;
            end

         // 4
         5'b00100:
            begin
                  HEX5 = hOff;
                  HEX4 = h4;
            end

         // 5
         5'b00101:
            begin
                  HEX5 = hOff;
                  HEX4 = h5;
            end

         // 6
         5'b00110:
            begin
```

```
77                      HEX5 = hOff;
78                      HEX4 = h6;
79                  end
80
81              // 7
82              5'b00111:
83                  begin
84                      HEX5 = hOff;
85                      HEX4 = h7;
86                  end
87
88              // 8
89              5'b01000:
90                  begin
91                      HEX5 = hOff;
92                      HEX4 = h8;
93                  end
94
95              // 9
96              5'b01001:
97                  begin
98                      HEX5 = hOff;
99                      HEX4 = h9;
100                 end
101
102             // A(10)
103             5'b01010:
104                 begin
105                     HEX5 = h1;
106                     HEX4 = h0;
107                 end
108
109             // B(11)
110             5'b01011:
111                 begin
112                     HEX5 = h1;
113                     HEX4 = h1;
114                 end
115
116             // C(12)
117             5'b01100:
118                 begin
119                     HEX5 = h1;
120                     HEX4 = h2;
121                 end
122
123             // D(13)
124             5'b01101:
125                 begin
126                     HEX5 = h1;
127                     HEX4 = h3;
128                 end
129
130             // E(14)
131             5'b01110:
132                 begin
133                     HEX5 = h1;
134                     HEX4 = h4;
135                 end
136
137             // F(15)
138             5'b01111:
139                 begin
140                     HEX5 = h1;
141                     HEX4 = h5;
142                 end
143
144             // 16
145             5'b10000:
146                 begin
147                     HEX5 = h1;
148                     HEX4 = h6;
149                 end
150
151             // 17
152             5'b10001:
```

```
153                 begin
154                         HEX5 = h1;
155                         HEX4 = h7;
156                 end
157
158             // 18
159             5'b10010:
160                 begin
161                         HEX5 = h1;
162                         HEX4 = h8;
163                 end
164
165             // 19
166             5'b10011:
167                 begin
168                         HEX5 = h1;
169                         HEX4 = h9;
170                 end
171
172             // 20
173             5'b10100:
174                 begin
175                         HEX5 = h2;
176                         HEX4 = h0;
177                 end
178
179             // 21
180             5'b10101:
181                 begin
182                         HEX5 = h2;
183                         HEX4 = h1;
184                 end
185
186             // 22
187             5'b10110:
188                 begin
189                         HEX5 = h2;
190                         HEX4 = h2;
191                 end
192
193             // 23
194             5'b10111:
195                 begin
196                         HEX5 = h2;
197                         HEX4 = h3;
198                 end
199
200             // 24
201             5'b11000:
202                 begin
203                         HEX5 = h2;
204                         HEX4 = h4;
205                 end
206
207             // 25
208             5'b11001:
209                 begin
210                         HEX5 = h2;
211                         HEX4 = h5;
212                 end
213
214             // 26
215             5'b11010:
216                 begin
217                         HEX5 = h2;
218                         HEX4 = h6;
219                 end
220
221             // 27
222             5'b11011:
223                 begin
224                         HEX5 = h2;
225                         HEX4 = h7;
226                 end
227
228             // 28
```

```
229             5'b11100:
230                begin
231                      HEX5 = h2;
232                      HEX4 = h8;
233                end
234
235             // 29
236             5'b11101:
237                begin
238                      HEX5 = h2;
239                      HEX4 = h9;
240                end
241
242             // 30
243             5'b11110:
244                begin
245                      HEX5 = h3;
246                      HEX4 = h0;
247                end
248
249             // 31
250             5'b11111:
251                begin
252                      HEX5 = h3;
253                      HEX4 = h1;
254                end
255          endcase
256       end
257    endmodule
258
259    module Hex7_testbench();
260       logic [6:0] HEX5, HEX4;
261       logic [4:0] count;
262
263       Hex7 dut (.count, .HEX5, .HEX4);
264
265       integer i;
266
267       initial begin
268             for (i=0; i<2**5; i++) begin
269                count = i; #10;
270             end
271          $stop; // end simulation
272       end
273    endmodule
```

```
1   /*
2   Mina Gao and Justin Sim
3   2/12/2024
4   EE 371 Hussein
5   Lab 4, Task 2
6
7      This module implements the FSM of the binary search, which manages the state transitions
    and
8      determines control signals base on the current state of the fsm.
9
10     inputs:
11        clk                  clock signal
12        reset                reset signal
13        start                start signal
14        MATCH                indicated if the value has been found
15        DNE                  indicated if the value has not been found (do not exist in the
    memory)
16        GT                   greater than, when target value >  q (the value read from the
    memory)
17        eq                   when target value is equal to q
18     outputs:
19        load_A               loading the register
20        inc                  updating searching bound (increase left bound)
21        dec                  updating searching bound (decrease right bound)
22  */
23  module binary_search_FSM (clk, reset, start, MATCH, DNE, GT, load_A, inc, dec,eq);
24     input logic clk, reset;
25     input logic start;
26     input logic MATCH, DNE, GT, eq;
27     output logic load_A, inc, dec;
28
29     /* three states
30        s0: idle
31        s1: search
32        s2: done
33     */
34     enum {s0, s1, s2} ps, ns;
35
36     always_comb begin
37        case (ps)
38           s0:   if (start)     ns = s1;
39                 else           ns = s0;
40
41           s1:   if (MATCH)     ns = s2;
42                 else if (DNE)  ns = s2;
43                 else           ns = s1;
44
45           s2:   if (start)     ns = s2;
46                 else           ns = s0;
47        endcase
48     end
49
50     always_ff @(posedge clk) begin
51        if (reset)   ps <= s0;
52        else         ps <= ns;
53     end
54
55     assign load_A = (ps == s0);
56     assign inc = ((ps == s1) & GT & !eq);
57     assign dec = ((ps == s1) & ~GT & !eq);
58
59  endmodule
60
61  module binary_search_FSM_tb ();
62     logic clk, reset, start;
63     logic MATCH, DNE, GT;
64     logic load_A, inc, dec;
65
66     binary_search_FSM dut(.*);
67
68     // clock setup
69     parameter clock_period = 100;
70
71     initial begin
72        clk <= 0;
73        forever #(clock_period / 2) clk <= ~clk;
```

```
74        end
75
76        initial begin
77           // reset and test inc
78           reset <= 1; start <= 0; MATCH <= 0; DNE <= 0; GT <= 0;      @(posedge clk);
79                                                                       @(posedge clk);
80           reset <= 0; start <= 1; MATCH <= 1;                repeat(3)@(posedge clk);
81
82           // reset and test abstract input
83           start <= 1; reset <= 1;                                    repeat(2)@(posedge clk);
84
85           // reset and test dec
86           reset <= 1; start <= 0; MATCH <= 0; DNE <= 0; GT <= 1;      @(posedge clk);
87                                                                       @(posedge clk);
88           reset <= 0; start <= 1; MATCH <= 1;                repeat(3)@(posedge clk);
89
90           $stop; // end simulation
91        end
92     endmodule
93
```

```
1   /*
2   Mina Gao and Justin Sim
3   2/12/2024
4   EE 371 Hussein
5   Lab 4, Task 2
6
7      This module implements the datapath of the binary search, which handles data
8      operations and interfacing with the RAM. Control signals outputed from binary
9      search FSM module guide the opreations of datapath.
10
11     inputs:
12        clk                  clock signal
13        [7:0] A_in           the value to be searched for in the memory
14        load_A, inc, dec     control signals
15     outputs:
16        [4:0] addr           address in the memory where the search operation reached
17        MATCH                indicated if the value has been found
18        DNE                  indicated if the value has not been found (do not exist in the
   memory)
19        GT                   greater than, when target value >  q (the value read from the
   memory)
20        DONE                 indicated if the search opreation is complete
21        eq                   when target value is equal to q
22        [7:0] q              the value retried from the memory
23
24  */
25  module binary_search_datapath  (clk, reset, A_in, load_A, inc, dec, addr, MATCH, DNE, GT,
   DONE, q, eq);
26     input logic clk, reset;
27     input logic [7:0] A_in;
28     input logic load_A, inc, dec;
29     output logic [4:0] addr;
30     output logic MATCH, DNE, GT, DONE, eq;
31     output logic [7:0] q;
32
33
34     logic [7:0] datatemp;            // unused logic for data port in ram
35     logic [4:0] wrtemp;              // unused logic for wr address port
36     logic [4:0] new_addr;
37
38     logic [4:0] MSB, LSB, MID, MID_temp;
39
40
41     always_ff @(posedge clk) begin
42           if (reset) begin
43              MSB <= 5'd31;
44              LSB <= 5'd0;
45           end else if (inc) begin
46              LSB <= MID + 1;
47           end else if (dec) begin
48              MSB <= MID - 1;
49           end else if (!inc & !dec) begin
50              MSB <= MSB;
51              LSB <= LSB;
52           end
53     end
54
55     assign new_addr = (MSB + LSB) / 2;
56
57     ram32x8 ram(
58     .clock(clk),
59     .data(datatemp),
60     .rdaddress(new_addr),
61     .wraddress(wrtemp),
62     .wren(0),
63     .q(q));              // 8'b output at the given address
64
65     always_ff @(posedge clk) begin
66        MID_temp <= new_addr;
67     end
68     always_ff @(posedge clk) begin
69        MID <= MID_temp;
70     end
71
72     assign MATCH = (A_in == q);
73     assign DNE = (MSB == LSB);
```

```
74      assign GT = (A_in > q);
75      assign eq = (A_in == q);
76      assign DONE = MATCH | (MSB == LSB);
77      assign addr = MID;
78
79  endmodule
80
81  `timescale 1 ps / 1 ps
82  module binary_search_datapath_tb  ();
83      logic clk, reset;
84      logic [7:0] A_in;
85      logic load_A, inc, dec;
86      logic [7:0] q;
87      logic [4:0] addr;
88      logic MATCH, DNE, GT, DONE, eq;
89
90      binary_search_datapath  dut(.*);
91
92      // clock setup
93      parameter clock_period = 100;
94
95      initial begin
96          clk <= 0;
97          forever #(clock_period / 2) clk <= ~clk;
98      end
99
100     initial begin
101         // Increment all the way up until MATCH
102         reset <= 1; load_A <= 0; A_in <= 8'd31;   @(posedge clk);
103                                                   @(posedge clk);
104         reset <= 0; load_A <= 1;                  @(posedge clk);
105                                                   @(posedge clk);
106         inc <= 1;                           repeat(15)@(posedge clk);
107         // Decrement all the way down until MATCH
108         reset <= 1; inc <= 0; load_A <= 0; A_in <= 8'd0;      @(posedge clk);
109                                                   @(posedge clk);
110         reset <= 0; load_A <= 1;                  @(posedge clk);
111                                                   @(posedge clk);
112         dec <= 1;                           repeat(15)@(posedge clk);
113
114         $stop; // end simulation
115
116     end
117 endmodule
118
```

```
/*
Mina Gao and Justin Sim
2/12/2024
EE 371 Hussein
Lab 4, Task 2

   This module implements a binary search algorithm, which searches through an array
   to locate an 8-bit value A specified using switches SW7-0 on DE1_SoC board.
   If value A is found in the memory, display the address of the data on HEX display
   Uses LEDR8 and LEDR9 to represent the result of the binary search, LEDR9 for 'Found'
   and LEDR8 for 'Not Found'.

   inputs:
      CLOCK_50             clock signal
      All SW               switches on DE1_SoC
         SW9: start signal
         SW7-0: specify the value A
      All KEY              keys on DE1_SoC
         KEY0: Reset signal
   outputs:
      [6:0] HEX            7'b HEX display on DE1_SoC
         HEX1-0: display the address of A if found
      [9:0] LEDR           LED on DE1_SoC
         LEDR9: found
         LEDR8: not found

*/
module DE1_SoC(CLOCK_50, KEY, SW, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR);
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [9:0] SW;
   input logic [3:0] KEY;
   input logic CLOCK_50;

   logic clk, reset, start;
   logic load_A, inc, dec;
   logic [7:0] A_in;
   logic [4:0] addr;
   logic MATCH, DNE, GT, eq, DONE;
   logic [7:0] q;                    // output value of a given RAM location
   logic [6:0] HEX_1, HEX_0;

   /* Turn off HEX2, HEX3, HEX4, HEX5 */
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

   ////////////////////////////////////////////////////////////////////
   /////// Define Intermediate Logic to output ports on DE1_SoC ///////
   assign clk = CLOCK_50;
   assign reset = ~KEY[0];
   assign start = SW[9];
   assign A_in = SW[7:0];

   // HEX0-1 will display location of A if exists,
   // if not, HEX0-1 will be turned off
   assign HEX1 = MATCH ? HEX_1 : 7'b1111111;
   assign HEX0 = MATCH ? HEX_0 : 7'b1111111;

   // LEDR 9 will glow when A input is found,
   // if not, LEDR 8 will glow
   assign LEDR[9] = MATCH ? 1'b1 : 1'b0;
   assign LEDR[8] = DNE ? 1'b1 : 1'b0;

   ////////////////////////////////////////////////////////////////////
   ////////////////////Module Instantiations//////////////////////////


   /* Displays the address of A on HEX 1, if found*/
   HexadecimalConverter disp_out_h1 (.clk(CLOCK_50), .binary_input({3'b000, addr[4]}), .
hexadecimal_output(HEX_1));

   /* Displays the address of A on HEX 0, if found*/
   HexadecimalConverter disp_out_h0 (.clk(CLOCK_50), .binary_input(addr[3:0]), .
hexadecimal_output(HEX_0));
```

```
75
76      //////////////Binary Search FSM/CONTROL Module://///////////////////
77      /*Inputs:
78         clk, reset
79         start        start algorithm
80         MATCH        indicates User input matches value at ram location
81         DNE          indicates User input does not exist in ram
82         GT           intermediate logic for binary search algo
83      Outputs:
84         load_A       control to set input
85         inc          control for algo to increment
86         dec          control for algo to decrement
87      */
88      binary_search_FSM ct (.*);
89
90
91      ////////////////Binary Search DATAPATH Module://///////////////////
92      /*Inputs:
93            clk, reset
94            [7:0] A_in      user Input on SW[7:0]
95            load_A          "
96            inc             "
97            dec             "
98      Outpts:
99            [4:0] addr      value of address algorithm's current iteration
100           MATCH           "
101           DNE             "
102           GT              "
103           DONE            Indicates Algorithm is finished wether or not FOUND input in ram
104     */
105     binary_search_datapath dp (.*);
106
107  endmodule                //DE1_SoC
108  //-------------------------------------------------------------------------------------------
     -------------
109  //DE1_SoC_testbench tests the expected and unexpected behaviors, reset behaviors
110
111  `timescale 1 ps / 1 ps
112  module DE1_SoC_tb();
113
114     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
115     logic [9:0] LEDR;
116     logic [9:0] SW;
117     logic [3:0] KEY;
118     logic CLOCK_50;
119
120     DE1_SoC dut(.*);
121
122     logic reset, start;
123     logic load_A, inc, dec;       // control signals for binary search algorithm
124     logic [7:0] A_in;             // User input: value to be searched for
125     logic [4:0] addr;             // Location/Address (if exists) of A in RAM
126     logic MATCH, DNE, GT, DONE;   // Intermediate Logic for datapath
127     logic [7:0] q;
128
129     // Assign Intermediate Logic to DE1_SoC ports
130     assign reset = ~KEY[0];
131     assign MATCH = LEDR[9];
132     assign DNE   = LEDR[8];
133     assign start = SW[9];
134     assign A_in  = SW[7:0];
135
136     ///// Testbench Clock Setup/////
137     parameter clock_period = 100;
138     initial begin
139        CLOCK_50 <= 0;
140        forever #(clock_period /2) CLOCK_50 <= ~CLOCK_50;
141     end
142
143     initial begin
144        // Expected Behavior: FOUND A_in = 5 @ ADDRESS 5
145
146        KEY[0] <= 0;       repeat(2)@(posedge CLOCK_50);// reset
147        KEY[0] <= 1;       repeat(2)@(posedge CLOCK_50);// reset
148
149        SW[7:0] <= 8'd21;     SW[9]<=1;       repeat(20)@(posedge CLOCK_50);
```

```
150
151         KEY[0] <= 0;       @(posedge CLOCK_50);// reset
152         KEY[0] <= 1;       @(posedge CLOCK_50);// reset
153
154         SW[7:0] <= 8'd31;     SW[9]<=1;       repeat(20)@(posedge CLOCK_50);
155
156         KEY[0] <= 0;       @(posedge CLOCK_50);// reset
157         KEY[0] <= 1;       @(posedge CLOCK_50);// reset
158         $stop; // end simulation
159
160       end   //initial
161   endmodule                 //DE1_SoC
```